UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tilen Pogačnik

Ogrodje za zaznavanje dihanja in predvajanje video posnetkov v okolju Unity

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2016

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tilen Pogačnik

Ogrodje za zaznavanje dihanja in predvajanje video posnetkov v okolju Unity

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Matija Marolt

Ljubljana, 2016

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Tilen Pogačnik, z vpisno številko **63120252**, sem avtor diplomskega dela z naslovom:

Ogrodje za zaznavanje dihanja in predvajanje video posnetkov v okolju Unity (angl. *A framework for breathing detection and video playback in Unity*)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matije Marolta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 1. september 2016 Podpis avtorja:

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomske naloge razvijte ogrodje, ki bo razvijalcem omogočalo razvoj aplikacij za dihalno vadbo v okolju Unity. Pri tem razvijte in ovrednotite algoritem za zaznavanje dihanja. V ogrodje vključite tudi orodje za predvajanje video posnetkov na mobilnih napravah, ki v navezavi z detektorjem dihanja omogoča nadzor predvajanja video posnetkov z dihanjem v mikrofon.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
FSM	finite-state machine	končni avtomat
FFT	fast Fourier transform	hitra Fourierova transformacija
PLB	Pursed Lip Breathing	izdih skozi priprta usta
XML	Extensible Markup Language	razširljivi označevalni jezik

Povzetek

Naslov: Ogrodje za zaznavanje dihanja in predvajanje video posnetkov v okolju Unity

Dihalna vadba s tehniko izdiha skozi priprta usta ali Pursed lip breathing je dihalna tehnika, ki se uporablja pri zdravljenju bolezni dihal in odpravljanju stresa. Glavni cilj diplomskega dela je razvoj ogrodja za igralni pogon Unity, ki razvijalcem omogoča hitro in preprosto izdelavo poljubnih interaktivnih aplikacij za dihalno vadbo z uporabo tehnike izdiha skozi priprta usta. V ta namen je bil razvit algoritem za zaznavanje dihanja, njegova učinkovitost pa je bila pokazana s testiranjem. Razvito je bilo tudi orodje za predvajanje video posnetkov na mobilnih napravah. Obe orodji sta bili nato povezani v celoto, ki nam omogoča nadzor predvajanja video posnetkov z dihanjem v mikrofon. Velik poudarek je na medsebojni neodvisnosti obeh orodij, tako da se lahko uporabljata tudi ločeno, odvisno od potreb ciljne aplikacije.

Ključne besede: Unity, zaznavanje dihanja, video predvajalnik, dihalna vadba

Abstract

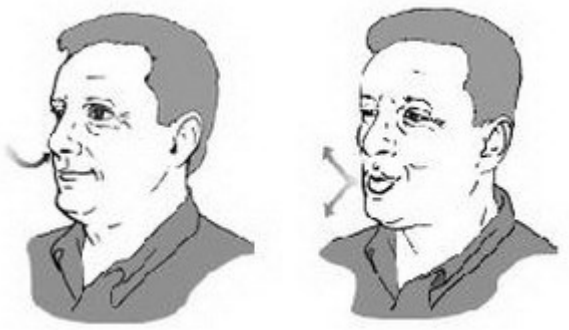
Title: A framework for breathing detection and video playback in Unity

Pursed lip breathing is a breathing technique that is used for treatment of respiratory diseases and stress reduction. The goal of this thesis is the development of a framework for Unity game engine that allows developers to quickly and easily produce interactive applications for respiratory training using pursed lip breathing technique. For this purpose, an algorithm for breathing detection was developed and its efficacy has been shown with testing. A tool for video playback on mobile devices was also developed. Both tools were then integrated together, to allow us to control video playback by breathing into the microphone. Great emphasis is on the mutual independence of the two tools so that they can also be used separately, depending on the needs of the target application.

Keywords: Unity, breathing detection, video player, breathing exercise

Poglavje 1 Uvod

Izdih skozi priprte ustnice (*angl. Pursed lip breathing*) ali PLB je dihalna tehnika, ki jo zdravniki in fizioterapevti priporočajo bolnikom, ki trpijo zaradi bolezni dihal, saj jim pomaga pri lajšanju simptoma oteženega dihanja oziroma dispneje [1]. Prav tako dihanje z uporabo PLB sprošča, pomirja in pomaga pri odpravljanju stresa ter tesnobe. Tehnika temelji na vdihu skozi nos in izvajanju podaljšanega izdiha čez priprte ustnice (Slika 1.1).



Slika 1.1 Dihalna tehnika PLB [2]

Čeprav je vadba dihanja z uporabo tehnike PLB uporabna za vsakogar, trenutno še ni splošno razširjena ali dobro poznana. Eden izmed možnih razlogov za to je, da čeprav živimo v svetu, kjer se s tehnologijo srečujemo na vsakem koraku, kvalitetnih aplikacij za dihalno vadbo skoraj ni na voljo. Da bi povečali poznavanje in razširjenost vadbe s tehniko PLB, smo poiskali način, da bi razvoj aplikacij za dihalno vadbo omogočili čim večjemu številu razvijalcev. Ugotovili smo, da specializiranih orodij za razvoj takih aplikacij še ni. Zato smo se odločili, da tako orodje razvijemo sami. Za implementacijo smo izbrali igralni pogon Unity, saj je trenutno zelo razširjen med razvijalci in omogoča razvoj raznovrstnih multimedijskih aplikacij za široko število različnih platform.

Razvili smo dve orodji: algoritem za zaznavanje dihanja in video predvajalnik, ki za nadzor predvajanja video posnetkov uporablja dihanje. Algoritem za zaznavanje dihanja je namenjen uporabi v kakršnemkoli projektu, kjer je potrebno zaznavanje dihanja, naj bo to le ena stopnja

v neki igri ali pa celotna aplikacija, namenjena dihalni vadbi. Zasnovan je tako, da deluje čim bolj robustno, deluje v realnem času, je preprost za uporabo in odporen proti govoru ter glasnemu okolju. Video predvajalnik ima integriran algoritem za zaznavanje dihanja in je namenjen hitremu razvoju video aplikacij. Razvijalcem omogoča nalaganje, predvajanje ter ustavljanje video posnetkov, preskakovanje na določen čas in menjanje različnih video posnetkov v realnem času. Velik poudarek je na preprostosti uporabe, tako da omogoča razvoj dihalnih video aplikacij brez potrebe po znanju programiranja ali izkušenj z delom v igralnem pogonu Unity.

V nadaljevanju bodo predstavljene uporabljene tehnologije, postopek razvoja in uporabe algoritma za zaznavanje dihanja, postopek razvoja in uporabe video predvajalnika in testiranje algoritma.

Poglavje 2 Uporabljene tehnologije

2.1 Unity

Unity [3] je igralni pogon (*angl. game engine*), ki je bil razvit s strani podjetja Unity Technologies leta 2005. Namenjen je razvoju 3D in 2D-iger, lahko pa se uporablja tudi v druge namene – medicinske vizualizacije, arhitekturne vizualizacije, interaktivne 3D-aplikacije in podobne namene.

Ena ključnih lastnosti pogona Unity je možnost razvijanja za več različnih platform hkrati. Ob izvozu (*angl. build*) se projekt namreč samodejno pretvori v želeno platformo, tako pa nam v večini primerov ni treba razvijati funkcionalnosti za vsako platformo posebej. Izvoz projektov je možen za naslednje platforme [4]:

- **mobilne platforme**: Android, iOS, Windows Phone, Tizen,
- **platforme za navidezno in obogateno resničnost**: Oculus Rift, Google Cardboard, Steam VR, Playstation VR, Gear VR, Microsoft Hololens,
- **namizne platforme**: Windows, Mac, Linux, Steam OS,
- **igralne konzole**: PlayStation 4, PlayStation Vita, Xbox One, Xbox 360, Wii U, Nintendo 3DS,
- **spletne platforme**: WebGL,
- **pametni TV-sprejemniki**: Android TV, Samsung Smart TV, tvOS.

Javnosti so na voljo tri različne verzije pogona Unity: zastonjska verzija Unity Free in plačljivi verziji Unity Plus ter Unity Pro. Unity Free vsebuje vse funkcionalnosti pogona, a ne omogoča spremembe uvodnega zaslona (*angl. splash screen*) z logotipom Unity (Slika 2.1). Uporaba Unity Free je mogoča, če ima razvijalec letni bruto dohodek, manjši od 100.000 ameriških dolarjev. Plačljivi verziji Unity Plus in Unity Pro prav tako vsebujeta vse funkcionalnosti pogona, dodatno pa omogočata spremembe uvodnega zaslona. Unity Plus stane 35 ameriških dolarjev mesečno in ga lahko uporabljajo razvijalci, ki imajo letni bruto dohodek, manjši od 200.000 ameriških dolarjev. Unity Pro stane 125 ameriških dolarjev mesečno in nima omejitve

letnega bruto dohodka. Prav tako obstaja verzija Unity Enterprise, ki je namenjena uporabi v večjih podjetjih in ima vse funkcionalnosti ostalih verzij, omogoča pa tudi dostop do izvorne kode pogona Unity. Cena Unity Enterprise ni javno objavljena [5].

Slika 2.1 Uvodni zaslon v pogonu Unity

Najnovejša izdaja pogona Unity je 5.4.0, ki je bila izdana 28. julija 2016. Med razvojem smo uporabljali nekoliko starejšo izdajo, in sicer 5.2.3, ker omogoča boljšo združljivost z uporabljenimi orodji PlayMaker (opisano v poglavju 2.2) in Easy Movie Texture (opisano v poglavju 2.3).

Uporabniški vmesnik pogona Unity (Slika 2.2) je v osnovi sestavljen iz šestih glavnih elementov [6]:

- **Projektno okno** (*angl. Project window*) služi za prikaz vseh datotek, ki so na voljo za uporabo v odprtem projektu. Znotraj projektnega okna lahko uvažamo in kreiramo nove datoteke.

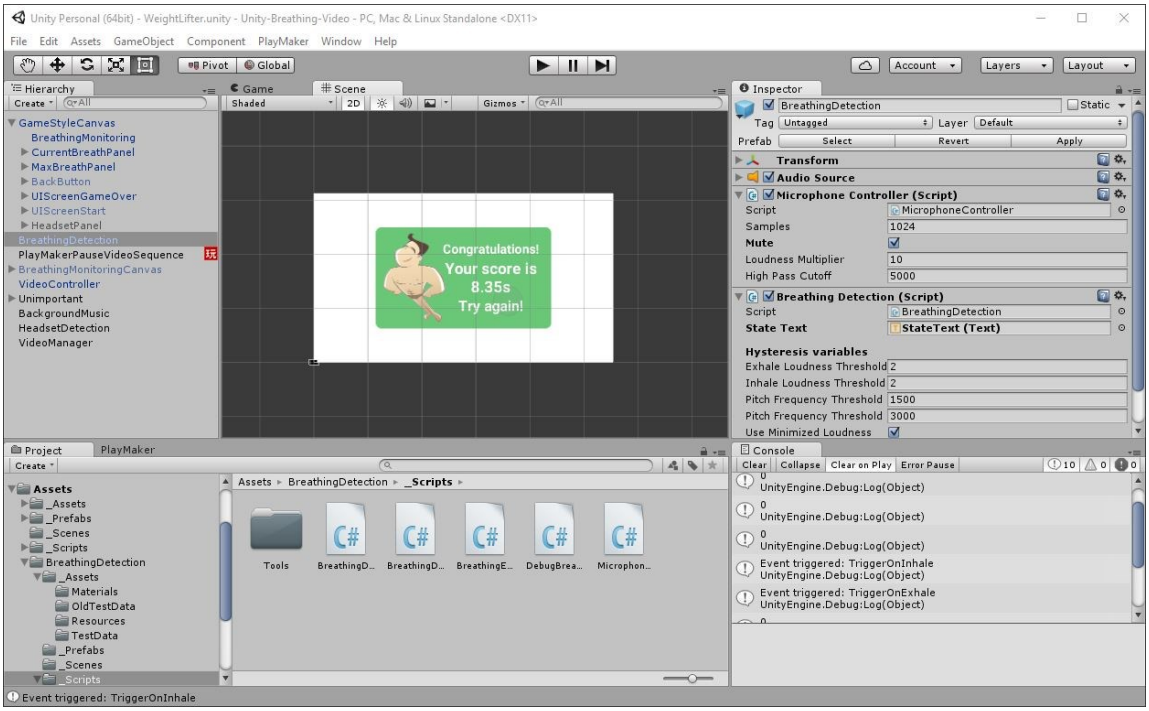
- **Pogled na sceno** (*angl. Scene view*) nam v obliki scenskega in igralnega pogleda prikaže trenutno sceno. Scenski pogled predstavlja interaktivno vizualizacijo trenutne scene v 3D ali 2D-pogledu in nam dovoljuje spreminjanje pozicije, orientacije ter velikosti objektov v prostoru. Igralni pogled (*angl. Game view*) prikazuje pogled na trenutno sceno skozi glavno kamero in simulira videz končne igre.

- **Hierarhija** (*angl. Hierarchy*) se uporablja za prikaz seznama trenutnih objektov v sceni in njihovih medsebojnih relacij.

– **Inšpektor** (*angl. Inspector*) služi za prikaz in urejanje vseh lastnosti trenutno izbranega objekta iz projektnega okna ali hierarhije. Prav tako nam omogoča dodajanje in odstranjevanje poljubnih komponent trenutno izbranega objekta.

– **Orodna vrstica** (*angl. Toolbar*) nudi dostop do osnovnih orodij za delo. Na levi strani nam omogoča izbiro orodij za manipuliranje z objekti v scenskem pogledu (spreminjanje pozicije, orientacije in velikosti). Na sredino so postavljeni gumbi za zagon in zaustavljanje igre v igralnem pogledu. Na desni strani nam omogoča dostop do storitev Unity Cloud Services in našega računa razvijalca.

– **Konzola** (angl. *Console*) nam prikazuje napake, opozorila in druga sporočila pogona Unity. Prav tako nam omogoča izpisovanje sporočil iz naših skript, kar nam lahko olajša postopek razhroščevanja projekta.



Slika 2.2 Uporabniški vmesnik pogona Unity

Razvoj v pogonu Unity temelji na objektih, poimenovanih *GameObject*. To so osnovni objekti brez lastnih funkcionalnosti. Imajo eno neodstranljivo komponento tipa *Transform*, ki hrani podatke o trenutni poziciji v prostoru, rotaciji in velikosti. Na vsak *GameObject* lahko pripenjamo komponente (*angl. Components*), ki implementirajo želene funkcionalnosti.

V pogon Unity je vgrajeno večje število različnih komponent, ki *GameObjectu* omogočajo izrisovanje (*angl. rendering*) na zaslon, uporabo fizike, razne vizualne efekte, animacijo, povezavo s spletom, oblikovanje uporabniškega vmesnika in predvajanje zvoka. Razvoj iger ni mogoč le z uporabo vgrajenih komponent, zato je večina razvoja usmerjena v razvoj lastnih komponent oziroma skript, ki komunicirajo z vgrajenimi komponentami pogona Unity.

Vsak *GameObject* lahko skupaj z vsemi njegovimi komponentami in parametri shranimo v projekt kot predlogo oziroma *Prefab* [7]. Tako si lahko shranimo objekte, ki so v aplikaciji uporabljeni večkrat in želimo, da imajo vsi enake komponente in njihove parametre. *Prefab*e nato v sceno dodajamo po principu povleci in spusti (*angl. drag-and-*drop). Prednost uporabe *Prefab*ov je v tem, da bodo vse kopije nekega *Prefab*a vedno imele enake vrednosti parametrov – če spremenimo katerokoli lastnost *Prefab*a, bo sprememba vidna na vseh njegovih instancah.

Za razvoj v pogonu Unity se uporabljajo programski jeziki C#, UnityScript (oziroma JavaScript za Unity) in Boo. Za potrebe diplomskega dela smo uporabljali programski jezik C#.

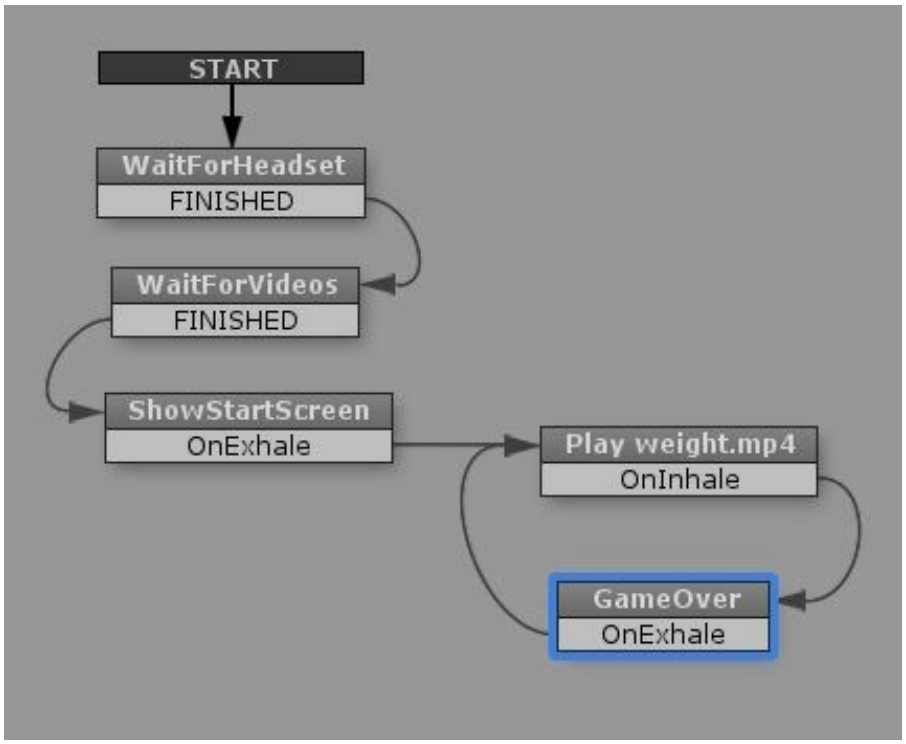
Za pomoč pri razvoju v pogonu Unity nam je na voljo tudi spletna trgovina Unity Asset Store [8]. Na spletni trgovini so objavljena sredstva (*angl. assets*), ki jih lahko uvozimo za uporabo v našem projektu. Trenutno je vse skupaj na voljo več kot 15.000 sredstev, ki vsebujejo vse od končanih projektov, animacij, 3D-modelov, senčilnikov (*angl. shaders*), tekstur in materialov do razvojnih orodij, razširitev uporabniškega vmesnika in novih komponent. Na voljo so tako plačljiva kot zastonjska sredstva. Na spletno trgovino lahko sredstva naloži kdorkoli, preden so javno objavljena, pa morajo prestati proces preverjanja. V našem projektu smo uporabili dve plačljivi sredstvi, in sicer Playmaker (predstavljeno v poglavju 2.2) in Easy Movie Texture (predstavljeno v poglavju 2.3).

2.2 Playmaker

Playmaker je orodje za vizualno programiranje z uporabo končnih avtomatov (*angl. Finite State Machine* ali FSM) znotraj pogona Unity. Omogoča nam, da program namesto s programsko kodo opišemo z grafičnimi elementi. Zato je primerno tako za manj izkušene razvijalce, ki želijo razvijati igre brez poglobljenega znanja programiranja, kot za izkušenejše razvijalce, ki jim omogoča lažjo in hitrejšo izdelavo prototipov iger.

Orodje Playmaker se uporablja tako, da želenemu objektu dodelimo končni avtomat Playmaker FSM in mu ročno določimo vsa možna stanja. Vsakemu stanju nato dodamo vnaprej definirane akcije, za katere želimo, da jih objekt izvaja, ko se nahaja v tem stanju. Na koncu pa določimo

še prehode med stanji, ki so definirani s pomočjo dogodkov (Slika 2.3). Dogodki, ki povzročijo menjavo stanja, se lahko prožijo kot rezultat izvajanja neke akcije znotraj avtomata Playmaker FSM ali pa pridejo od zunaj – torej iz drugih skript.



Slika 2.3 Primer avtomata Playmaker FSM

Orodje Playmaker ima vgrajenih precej osnovnih akcij, ki zadostijo večini potreb za razvoj preprostih iger. Poleg osnovnih akcij pa nam omogoča implementacijo lastnih akcij in posledično integracijo poljubne programske kode v lažje razumljiv format. Na sliki 2.4 je prikazano stanje »Play weight.mp4«, ki uporablja naše lastne akcije za nadzor predvajanja video posnetka, skrivanje elementov uporabniškega vmesnika in predvajanje zvočnih efektov.

Slika 2.4 Primer akcij na stanju "Play weight.mp4"

Orodje Playmaker je bilo razvito s strani podjetja Hutong Games in je na voljo v spletni trgovini Unity Asset Store [9]. Orodje je plačljivo in se prodaja po ceni 65 ameriških dolarjev. V primeru uporabe orodja Playmaker v skupinskih projektih je zahtevano, da ima vsak razvijalec svojo licenco.

2.3 Easy Movie Texture

Unity ima vgrajeno komponento MovieTexture [10], ki omogoča predvajanje in upravljanje poljubnih video posnetkov na vseh platformah z izjemo platform Android in iOS. Video posnetki se ob uvozu v projekt samodejno pretvorijo v format Ogg Theora, ki je edini podprt format za predvajanje z uporabo komponente MovieTexture.

Na platformah Android in iOS je v pogonu Unity predvajanje video posnetkov podprto le z uporabo funkcije Handheld.PlayFullScreenMovie [11]. Za našo aplikacijo to ni primerno, ker se pogon Unity med predvajanjem video posnetka zaustavi, posledično pa izgubimo ves nadzor nad predvajanjem. Prav tako nimamo možnosti prikaza uporabniškega vmesnika nad video posnetkom.

Na spletni trgovini Unity Asset Store je na voljo nekaj različnih orodij, ki implementirajo funkcionalnost prej omenjenega MovieTexture na platformah Android in iOS:

– Pop Movie Texture (cena 300.00 $),

– MPMP – Multi Platform Media Player (cena 175.00 $),

– Mobile Movie Texture (cena 75.00 $),

– Universal Media Player (cena 45.00 $),

– Easy Movie Texture (cena 65.00 $).

Po pregledu prej naštetih orodij smo se odločili za uporabo orodja Easy Movie Texture [12], saj je cenovno ugodno, ima vse funkcionalnosti, ki jih potrebujemo, deluje zanesljivo, prav tako pa zraven dobimo še vpogled v izvorno kodo.

Easy Movie Texture za svoje delovanje uporablja vtičnik (*angl. native plugin*), ki na napravi dekodira video posnetek, vsako sličico v obliki teksture shrani in pogonu Unity pošlje kazalec (*angl. pointer*) na mesto v spominu, kjer je shranjena. V pogonu Unity se ta tekstura nato prebere in izriše preko komponente Quad [13]. Vsakič, ko je na voljo nova sličica video posnetka, vtičnik pogonu Unity sporoči, naj posodobi teksturo in jo ponovno izriše. Uporaba orodja Easy Movie Texture za predvajanje video posnetkov je bolj natančno opisana v poglavju 4.1.

2.4 Breathing+ Headset

Za razvoj algoritma za zaznavanje dihanja smo uporabili slušalke z mikrofonom Breathing+ Headset (Slika 2.5).

Slika 2.5 Breathing+ Headset [14]

Ohišje je izdelano iz mehkega polietilena (*angl. polyethylene plastic*) in je po dolžini nastavljivo za različne velikosti glav. Med uporabo fizični kontakt med uporabnikom in ohišjem slušalk z mikrofonom ni potreben. Za optimalno zaznavanje diha omogoča stabilno pozicioniranje mikrofona v razdalji med enim in dvema centimetroma pred uporabnikovimi ustnicami. Mikrofon je zavarovan pred vlago v izdihanem zraku, prav tako pa so celotne slušalke z mikrofonom odporne proti pranju.

Za povezavo z računalnikom ali mobilnim telefonom je uporabljen 3.5 mm TRRS-priključek, ki omogoča istočasno delovanje vgrajenega mikrofona in slušalk z uporabo enega vtikača. V primeru, da želena naprava ne podpira uporabe TRRS-priključkov, je priložen adapter iz standarda TRRS na TRS. V primeru uporabe adapterja je omogočeno le še delovanje vgrajenega mikrofona.

Poglavje 3 Razvoj algoritma za zaznavanje dihanja

V tem poglavju bomo podrobno opisali postopek razvoja algoritma za zaznavanje dihanja. Opisali bomo postopek zajemanja podatkov iz mikrofona in način zaznavanja dihanja z izračunom energije zvoka. Nato bomo predstavili pomanjkljivosti algoritma in uporabljene pristope za njegovo izboljšavo. Predstavili bomo končen algoritem z vsemi izboljšavami, nato pa bomo podali še primer uporabe algoritma v poljubni aplikaciji.

3.1 Zajemanje podatkov iz mikrofona in izračun glasnosti

V pogonu Unity lahko obdelujemo le zvok, ki se predvaja prek komponente AudioSource [15], zato je bila naša prva naloga, da signal iz mikrofona povežemo s to komponento. Mikrofon snema posnetke, dolge eno sekundo, in jih shrani v obliko AudioClip [16], ki jo predvaja komponenta AudioSource. Vsako sekundo zavržemo posnetek prejšnje sekunde in začnemo istočasno snemati ter predvajati naslednji posnetek (Slika 3.1). Tako delovnega pomnilnika ne obremenimo preveč, saj imamo vedno shranjen le en posnetek, ki je dolg največ eno sekundo.

```
void prepareMicrophone(){
        if (Microphone.devices.Length > 0) {

            //Gets the maxFrequency and minFrequency of the microphone
            Microphone.GetDeviceCaps (Microphone.devices [0], out minFrequency, out maxFrequency);

            //Start microphone recording and set it as the AudioClip on the AudioSource
            aSource.clip = Microphone.Start (Microphone.devices [0], true, 1, maxFrequency);
            aSource.loop = true;

            //Wait until microphone starts recording
            while (!(Microphone.GetPosition(Microphone.devices[0]) > 0)) {
            }

            //Start playing the recording in real-time
            aSource.Play();
            isMicrophoneReady = true;

        } else {
                Debug.LogWarning("No microphone detected.");
        }
    }
```

Slika 3.1 Inicializacija mikrofona

Podatke trenutno predvajanega zvoka lahko pridobimo z uporabo naslednjih funkcij:

- **AudioSource.GetOutputData** (float[] samples, int channel)
    - Funkcija v tabelo *samples* vstavi zadnjih *n* zvočnih vzorcev, kjer *n* predstavlja dolžino tabele *samples*.
    - Parameter *channel* določi, iz katerega kanala nam bo funkcija vrnila vrednosti. Za naše potrebe smo vedno uporabljali kanal 0.
- **AudioSource.GetSpectrumData** (float[] samples, int channel, FFTWindow window)
    - Funkcija v tabelo *samples* vstavi rezultate hitre Fourierove transformacije trenutno predvajanega zvoka.
    - Parameter *channel* določi, iz katerega kanala nam bo funkcija vrnila vrednosti. Za naše potrebe smo vedno uporabljali kanal 0.
    - Parameter *window* določi, katero okensko funkcijo bomo uporabljali za izračune hitre Fourierove transformacije.

V večini primerov uporabe algoritma želimo, da se zvok iz mikrofona ne predvaja nazaj preko zvočnikov. Komponenta AudioSource ima funkcionalnost utišanja in spreminjanja glasnosti predvajanja, a ti funkcionalnosti za naš algoritem nista primerni. V primeru utišanja komponente AudioSource ne moremo več ničesar računati – prej omenjeni funkciji AudioSource.GetOutputData in AudioSource.GetSpectrumData nam na utišani komponenti AudioSource vedno vrneta ničle. Zato moramo izhod komponente AudioSource povezati z mešalnikom zvoka AudioMixer [17], ki nam omogoča utišanje zvoka brez spreminjanja komponente AudioSource.

Ko imamo pripravljen mikrofon, lahko začnemo z računanjem glasnosti signala mikrofona. Za izračun uporabljamo povprečne vrednosti korena vsote uteženih kvadratov (*angl. root mean square*), ki ga izračunamo po enačbi (3.1). Izračun glasnosti se nahaja znotraj funkcije FixedUpdate, ki jo pogon Unity samodejno pokliče vsakih 20 ms, neodvisno od hitrosti sličic (*angl. frame rate*), s katero deluje aplikacija. Tako petdesetkrat na sekundo izračunamo novo glasnost signala mikrofona.

Dihanje lahko zaznavamo že samo z uporabo glasnosti – če smo v stanju vdiha in je glasnost večja od ročno določenega praga, potem spremenimo trenutno stanje dihanja v stanje izdiha in sprožimo ustrezen dogodek. Enako velja za nasproten primer – če smo v stanju izdiha in je

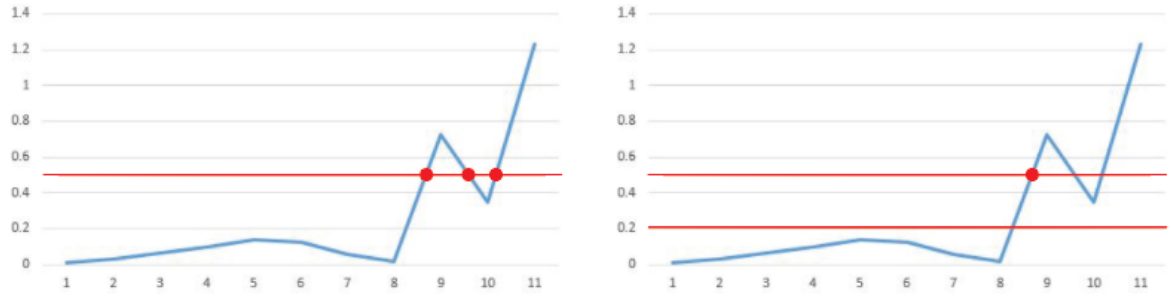
glasnost manjša od ročno določenega praga, potem spremenimo trenutno stanje dihanja v stanje vdiha in sprožimo ustrezen dogodek. Tak algoritem sicer deluje, ampak je izredno občutljiv na zvok v ozadju in ni odporen proti govoru, saj ga zazna kot dihanje. Pogosto prag prečka večkrat in namesto enega izdiha oziroma vdiha zazna več izdihov in vdihov.

$$Glasnost = \sqrt{\frac{\sum_{n=1}^{Length(Samples)} Samples[n]^2}{Length(Samples)}} \tag{3.1}$$

3.2 Dvojni prag

Mikrofon izdih zazna kot šum, zato se glasnost ne spreminja enakomerno – graf glasnosti je nazobčan. Zato se lahko zgodi, da glasnost signala mikrofona ob začetku ali koncu izdiha prag prečka večkrat, kar pomeni, da se stanje dihanja spremeni večkrat, kot bi se moralo. To težavo bi lahko rešili tako, da bi zavrgli vse izdihe, ki so krajši od nekega vnaprej definiranega trajanja, a smo zaradi potrebe po delovanju algoritma v realnem času morali poiskati drugačno rešitev.

Uvedli smo uporabo dvojnega praga – namesto uporabe enega samega praga, ki služi kot meja med vdihom in izdihom, uporabimo dva različna praga. Višji prag služi kot meja za izdih, nižji prag pa služi kot meja za vdih. Tako se rešimo prej opisane težave večkratnega spreminjanja stanja dihanja, hkrati pa pridobimo večji nadzor nad zaznavanjem dihanja in manjšo občutljivost na govor in zvok v ozadju. Razlika med enojnim in dvojnim pragom pri zaznavanju začetka izdiha je vidna na Slika 3.2.

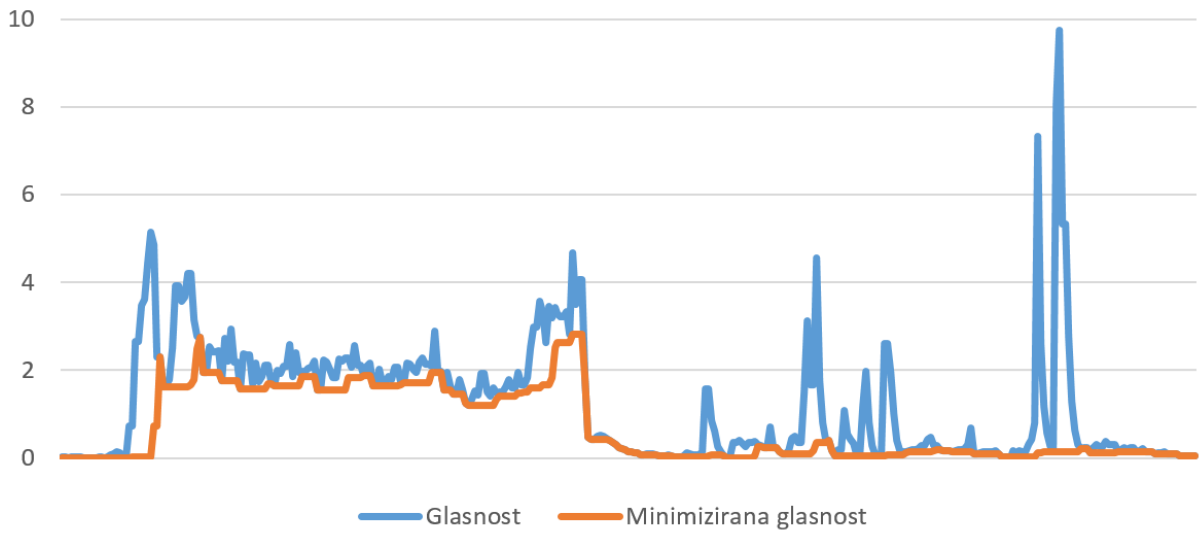


Slika 3.2 Primerjava zaznavanja algoritma z enojnim pragom (levo) in algoritma z dvojnim pragom (desno)

3.3 Problem presluha

Presluh (*angl. crosstalk*) je motilni signal, ki nastane zaradi medsebojnega vpliva sosednjih povezovalnih linij. Elektromagnetno polje, ki ga povzroči signal v eni liniji, vpliva na sosednje linije in v njih povzroči motnjo signala [18]. V našem primeru se ob predvajanju glasbe na slušalkah Breathing+ Headset pojavi motilni signal na mikrofonski liniji, posledično pa se lahko pojavijo napačna zaznavanja.

V našem algoritmu smo problem presluha reševali na dva načina. Mikrofon Breathing+ Headset ima sam po sebi dobro odpornost na zvok v ozadju, zato smo do sedaj lahko predpostavljali, da bo izračunana glasnost mikrofona med vdihom blizu ničle. Ko pa se pojavi presluh, tega ne moremo več predpostavljati, saj bo motnja zaradi presluha precej bolj opazna kot kakršenkoli zvok v ozadju, posledično pa bo izračunana glasnost mikrofona večja. Za zaznavanje začetka izdiha to ne predstavlja problema, saj je motnja zaradi presluha precej manjša od pragu glasnosti za začetek izdiha. Težava se pojavi pri zaznavanju konca izdiha, saj ne moremo zagotoviti, da se bo izračunana glasnost mikrofona še kdaj vrnila v bližino ničle. Zaradi tega smo morali povišati prag glasnosti za konec izdiha. Tako odpravimo težavo presluha, a zmanjšamo robustnost algoritma. Nov prag glasnosti za konec izdiha mora biti namreč postavljen tako visoko, da se pojavi možnost, da algoritem konec izdiha zazna prehitro. Zato smo morali poiskati še način zmanjšanja vpliva presluha na izračunano glasnost mikrofona. Opazili smo, da se med vdihom motnja glasnosti zaradi presluha spreminja zelo hitro – hitro doseže vrh, prav tako pa se hitro vrne nazaj na nizko vrednost. Zato smo v algoritem dodali pomnjenje zgodovine poljubnega števila zadnjih izračunanih glasnosti in za zaznavanje vdihov ter izdihov pričeli uporabljati najmanjšo glasnost v zgodovini izračunanih glasnosti. Posledično lahko prag glasnosti za konec izdiha spustimo nižje, kjer ni več nevarnosti, da bi algoritem prehitro zaznal konec izdiha. Ker se računanje glasnosti izvaja na vsakih 20 ms, pride zaradi minimizacije glasnosti do latence pri zaznavanju začetka izdiha. S krajšanjem zgodovine pomnjenja glasnosti latenca pada, hkrati pa pada tudi odpornost na presluh in govor. Z daljšanjem zgodovine pomnjenja glasnosti pa odpornost na presluh in govor narašča, hkrati pa se veča tudi latenca. Privzeto pomnimo zadnjih deset izračunanih glasnosti, kar nam omogoča dobro odpornost na presluh in govor, povzroča pa latenco 200 ms, ki je še vedno dovolj nizka za hitro zaznavanje izdihov. Na sliki 3.3 je prikazan graf izračunane glasnosti pred in po minimizaciji. Leva polovica grafa predstavlja izdih, desna pa govor.



Slika 3.3 Graf izračunane glasnosti pred in po minimizaciji

3.4 Frekvenčna analiza signala

S frekvenčno analizo signala želimo povečati odpornost algoritma na napačno zaznavanje izdihov med govorom. Čeprav je minimizacija glasnosti že precej povečala odpornost na govor, so se še vedno pojavljale napačne zaznave. Unity nam omogoča, da iz trenutno predvajajočega se zvoka z uporabo hitre Fourierove transformacije oziroma FFT razmeroma preprosto pridobimo podatke o trenutnih frekvencah signala. Za izračun FFT moramo definirati tudi okensko funkcijo, ki se uporablja za zmanjševanje učinka spektralnega puščanja (*angl. spectral leakage*) [19]. Za potrebe našega algoritma smo uporabili okensko funkcijo Blackman-Harris.

Rezultate FFT smo poskušali analizirati na več načinov: računanje centroida frekvenc, računanje vsote in primerjanje oblike. Noben način se ni izkazal za uporabnega – ali ni pokazal bistvene razlike med govorom in izdihom ali pa je bil preveč podoben že izračunani glasnosti mikrofona.

Kasneje smo ugotovili, da si lahko pomagamo z iskanjem maksimuma med rezultati algoritma FFT in preverimo, ali se nahaja znotraj nekega vnaprej določenega intervala. Ugotovili smo namreč, da je najmočnejša frekvenca v spektru med govorom v veliki večini primerov nižja od 500 Hz. Izjemo predstavljajo glasovi Č, S in Ž, ki imajo pogosto precej višje frekvence od normalnega govora. To nam ne predstavlja večjih težav, saj so njihove frekvence višje od izdihov, pri katerih frekvenca maksimuma rezultatov algoritma FFT niha med 1000 Hz in 3000 Hz.

Za začetek izdiha smo tako določili dodaten pogoj, da mora biti frekvenca maksimuma rezultatov algoritma znotraj vnaprej določenega intervala. Dodatnega pogoja za konec izdiha nismo definirali.

Z združitvijo vseh metod, opisanih v poglavjih 3.1 do 3.4, smo prišli do končne verzije algoritma, ki je s psevdokodo prikazana na sliki 3.4.

```
Stanje = vdih;
while(true){
    Izračunaj minimizirano glasnost;
    Izračunaj maksimum FFT;

    if (stanje == vdih){
        if (minimizirana glasnost večja od praga za začetek izdiha
            && maksimum FFT večji od 1000 Hz && maksimum FFT manjši od 3000 Hz){
            Stanje = izdih;
            Sproži dogodek za začetek izdiha;
        }
    } else {
        if (minimizirana glasnost manjša od praga za konec izdiha){
            Stanje = vdih;
            Sproži dogodek za konec izdiha;
        }
    }
    Počakaj 20ms;
}
```

Slika 3.4 Psevdokoda končnega algoritma za zaznavanje dihanja

3.5 Uporaba algoritma za zaznavanje dihanja

Razvit algoritem ni namenjen zgolj za uporabo v eni, točno določeni aplikaciji, ampak je razvit kot orodje, s katerim lahko drugi razvijalci v svoj Unity projekt preprosto dodajo funkcionalnost zaznavanja dihanja. Zato nam je bilo zelo pomembno, da je razvit algoritem popolnoma neodvisen od drugih komponent v projektu. Uporabnik lahko algoritem v svoj projekt uvozi z uporabo Unity paketa (*angl. Unitypackage*). Nato mora v hierarhijo scene, kjer želi zaznavati dihanje, z vlečenjem (*ang. drag-and-drop*) vstaviti priložen *Prefab* z imenom BreathingDetection in algoritem bo pripravljen za uporabo.

Ker mnogo razvijalcev uporablja orodje Playmaker, smo algoritem povezali tudi z njim. Za delovanje z orodjem Playmaker mora razvijalec na objekt, ki vsebuje komponento Playmaker FSM, dodati še našo komponento BreathingDetectionPlaymakerEvents, ki ob zaznanem vdihu oziroma izdihu sproži ustrezen dogodek na komponenti Playmaker FSM znotraj tega objekta. Uporaba algoritma brez uporabe orodja Playmaker je prav tako mogoča. V takem primeru se

moramo iz komponent, kjer želimo zaznavati dihanje, ročno naročiti na dogodke algoritma za zaznavanje dihanja in definirati funkcije, ki se bodo izvedle ob vsaki spremembi stanja dihanja (Slika 3.5).

```
void OnEnable(){
    BreathingEvents.onExhale += exampleExhaleFunction;
    BreathingEvents.onInhale += exampleInhaleFunction;
}

void OnDisable(){
    BreathingEvents.onExhale -= exampleExhaleFunction;
    BreathingEvents.onInhale -= exampleInhaleFunction;
}

void exampleExhaleFunction(){
    //This will be called when BreathingEvents.onExhale is triggered
}
void exampleInhaleFunction(){
    //This will be called when BreathingEvents.onInhale is triggered
}
```

Slika 3.5 Naročanje na dogodke iz poljubne skripte

Naš algoritem je optimiziran za delovanje na mikrofonu Breathing+ Headset. Za bolj napredne uporabnike so izpostavljeni vsi parametri, ki se uporabljajo v algoritmu. Tako je razvijalcem omogočeno, da algoritem priredijo svojim potrebam in ga optimizirajo za delovanje na želenih napravah.

Poglavje 4 Razvoj predvajalnika video posnetkov

V tem poglavju bomo podrobno opisali razvoj orodja za predvajanje video posnetkov v pogonu Unity. Predstavili bomo postopek razvoja knjižnice za nadzor predvajanja video posnetkov z uporabo orodja Easy Movie Texture. Pojasnili bomo povezavo prej omenjene knjižnice z orodjem Playmaker in zaznavanjem dihanja. Opisali bomo postopek oblikovanja uporabniškega vmesnika, na koncu pa bomo še pregledali, kako so posamezni elementi video predvajalnika povezani med seboj in predstavili njihov način medsebojnega komuniciranja.

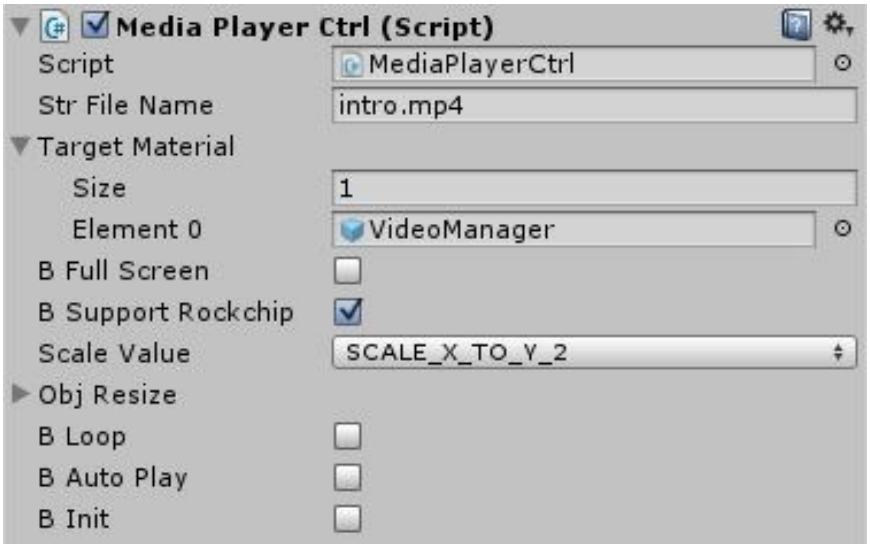
4.1 Razvoj knjižnice za nadzor predvajanja video posnetkov

Ker pogon Unity ne omogoča predvajanja video posnetkov na mobilnih napravah, smo se odločili za uporabo orodja Easy Movie Texture, ki nam predvajanje video posnetkov omogoči tudi na platformah Android in iOS. Splošna predstavitev orodja Easy Movie Texture se nahaja v poglavju 2.3.

V orodju Easy Movie Texture je priložena skripta MediaPlayerCtrl.cs, ki se uporablja kot komponenta in nam omogoča predvajanje video posnetkov. Javno ima izpostavljene naslednje parametre (Slika 4.1):

- **Str File Name**: Ime video posnetka, ki ga želimo predvajati. Če se posnetek ne nahaja v mapi StreamingAssets, moramo vnesti absolutno pot do posnetka.
- **Target Material**: Ciljni material, na katerega se bo izrisoval video posnetek v obliki tekstura.
- **B Full Screen**: Stikalo, s katerim naj bi imeli možnost predvajanja video posnetka v celozaslonskem načinu. Kasneje smo ugotovili, da predvajanje v celozaslonskem načinu z uporabo tega stikala deluje le na napravah s slikovnim razmerjem 16:9.
- **B Support Rockchip**: Naprave, ki uporabljajo nabor čipov Rockchip, podpirajo le 16-bitni video medpomnilnik (*angl. buffer*) in ne podpirajo neposrednega dostopa do mape StreamingAsset. Z uporabo tega stikala omogočimo predvajanje video posnetkov tudi na teh napravah, a je posledično kvaliteta predvajanja slabša.

- **Scale Value**: V primeru celozaslonskega načina predvajanja z uporabo tega parametra določimo, po kateri koordinatni osi naj se spreminja velikost videa.
- **B Loop**: Stikalo, s katerim določimo ponavljanje video posnetka.
- **B Auto Play**: Stikalo, s katerim določimo, če se posnetek začne predvajati takoj, ko se naloži.
- **B Init**: V dokumentaciji ni nobene omembe tega stikala. Predvajanje video posnetkov deluje neodvisno od tega, ali je to stikalo vključeno ali ne.



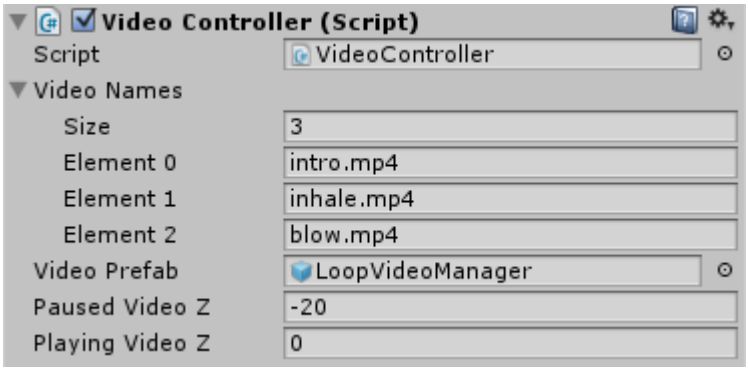
Slika 4.1 Pogled na skripto Media Player Ctrl v inšpektorju

Orodje Easy Movie Texture je trenutno najbolj priljubljeno orodje za predvajanje video posnetkov na mobilnih napravah v pogonu Unity, a vseeno vsebuje precej pomanjkljivosti. Orodje je izredno občutljivo na sinhronizacijo klicev programskega vmesnika – celotno orodje preneha delovati, če poskusimo na primer predvajati video posnetek, preden je pripravljen na predvajanje ali če želimo predvajati posnetek, ki se že predvaja. Uporabnikov prav tako ne opozori na morebitne napake pri uporabi programskega vmesnika, kar lahko povzroča velike težave uporabnikom, ki ne poznajo vseh podrobnosti uporabe tega orodja. Prav tako orodje ne omogoča enostavnega in hitrega preklapljanja med različnimi video posnetki.

Zaradi zgoraj omenjenih razlogov smo se odločili, da bomo razvili svojo knjižnico za komunikacijo z orodjem Easy Movie Texture, ki bo preprosta za uporabo, neobčutljiva na sinhronizacijo klicev, bo omogočala preklapljanje med različnimi video posnetki in bo uporabnika obveščala o kakršnihkoli napakah pri uporabi knjižnice.

Knjižnico smo poimenovali VideoController in ima izpostavljene naslednje parametre (Slika 4.2):

- **Video Names**: Tabela z imeni vseh video posnetkov, ki jih želimo uporabljati v aplikaciji. Pomembno je, da se vsi video posnetki nahajajo znotraj Unity projekta v mapi StreamingAssets. V nasprotnem primeru jih Unity ne bo vključil v izvoženo aplikacijo.
- **Video Prefab**: *Prefab* z vnaprej konfigurirano komponento MediaPlayerCtrl. VideoController to predlogo uporabi za generacijo objektov za vsak video, ki ga želimo predvajati v aplikaciji.
- **Paused Video Z**: Določi pozicijo objekta z video posnetkom na osi z, medtem ko se video posnetek ne predvaja in je uporabljen za skrivanje video posnetka.
- **Playing Video Z**: Določi pozicijo objekta z video posnetkom na osi z med predvajanjem video posnetka.



Slika 4.2 Pogled na skripto VideoController v inšpektorju

Ob zagonu aplikacije se za vsak video posnetek, ki je naveden v tabeli Video Names, generira svoj objekt po predlogi, določeni v polju Video Prefab. Predloga vsebuje prej omenjeno komponento MediaPlayerCtrl in izrisovalnik mreže z obliko Quad. V vsakem generiranem objektu se sproži nalaganje pripadajočega video posnetka. Med nalaganjem vsak video posnetek razširimo v celozaslonski način – če je širina slikovnega razmerja video posnetka večja od širine slikovnega razmerja zaslona, izenačimo širino video posnetka s širino zaslona in izračunamo novo višino video posnetka, da ohranimo pravilno slikovno razmerje. V nasprotnem primeru pa izenačimo višino video posnetka z višino zaslona in izračunamo novo širino video posnetka. Prav tako popravimo lego video posnetka v prostoru, da ni viden na zaslonu. Ko sta generacija in nalaganje video posnetkov dokončana, sprožimo dogodek, ki

uporabnika obvesti, da so video posnetki pripravljeni na uporabo. V primeru, da uporabnik pred tem poskuša predvajati kakšen nepripravljen video posnetek, ga na to opozorimo, a želen posnetek vseeno predvajamo takoj, ko je pripravljen. Med izvajanjem hranimo reference na nazadnje predvajan video posnetek in status nalaganja vsakega video posnetka.

Knjižnica ima izpostavljene naslednje javne funkcije, s katerimi lahko uporabniki nadzorujejo predvajanje video posnetkov:

- **PlayVideo** (string name, bool seekToStart, bool loop = false, bool hidePreviousVideo = true)
    - Funkcija poskuša predvajati video posnetek z imenom, podanim v parametru name. Spremeni mu pozicijo v prostoru, da je viden na zaslonu. Če obstaja nazadnje predvajan video posnetek, ga ustavi in skrije.
    - V primeru, da video posnetek s podanim imenom ne obstaja ali če še ni pripravljen na predvajanje, potem uporabnika obvesti o napaki.
- **PlayCurrentVideo** ()
    - V primeru, da obstaja neki nazadnje predvajan video posnetek, ga funkcija predvaja. Če tak posnetek ne obstaja, potem uporabnika obvesti o napaki.
- **PauseCurrentVideo** (bool hideVideoCanvas = true)
    - V primeru, da obstaja neki posnetek, ki se trenutno predvaja, ga funkcija pavzira. Če tak posnetek ne obstaja, potem uporabnika obvesti o napaki.
    - Če je parameter hideVideoCanvas enak true, potem tudi skrije objekt z video posnetkom.
- **StopCurrentVideo** (bool hideVideoCanvas = true)
    - Deluje podobno kot funkcija PauseCurrentVideo, le da video posnetek prevrti na začetek.
- **SeekCurrentVideo** (float seconds)
    - Trenutno predvajan video posnetek prevrti na čas, določen v sekundah s parametrom seconds.

– **SetVideoOnEnd** (string name, MediaPlayerCtrl.VideoEnd function)

  - Funkcijo, ki je določena s parametrom function, naroči na dogodek OnEnd podanega video posnetka, ki se sproži ob končanem predvajanju video posnetka.

  - V primeru, da video posnetek s podanim imenom ne obstaja ali če še ni pripravljen na predvajanje, potem uporabnika obvesti o napaki.

– **RemoveVideoOnEnd** (string name, MediaPlayerCtrl.VideoEnd function)

  - Podanemu video posnetku odstrani funkcijo, določeno s parametrom function, če je bila naročena na dogodek OnEnd. V primeru, da parametra function ne podamo, potem funkcija odstrani vse naročnike na dogodek OnEnd podanega video posnetka.

  - V primeru, da video posnetek s podanim imenom ne obstaja ali če še ni pripravljen na predvajanje, potem uporabnika obvesti o napaki.

V primeru napake vse naštete funkcije vrnejo false, drugače vrnejo true.

4.2 Podpora orodju Playmaker

Ker nam je preprostost uporabe našega orodja za predvajanje video posnetkov zelo pomembna, smo se odločili da razvito knjižnico za predvajanje video posnetkov in algoritem za zaznavanje dihanja povežemo z orodjem Playmaker. Tako omogočimo uporabnikom, ki niso vešči programiranja in dela s pogonom Unity, da v nekaj minutah izdelajo lastno aplikacijo za dihalno vadbo.

Slika 4.3 Naše Playmaker Akcije

Za povezavo s knjižnico smo definirali serijo lastnih akcij (Slika 4.3), ki komunicirajo s knjižnico za nadzor predvajanja video posnetkov. Poleg tega smo definirali še nekaj dodatnih akcij, ki niso povezane s knjižnico, ampak menimo, da so uporabne pri razvoju celotne aplikacije. Te akcije so:

- **Wait For Videos To Load**: Akcija počaka, da so vsi video posnetki naloženi in pripravljeni za predvajanje. Ob koncu izvajanja sproži vgrajen dogodek orodja Playmaker FINISHED, ki omogoči prehod na naslednje stanje avtomata Playmaker FSM. Predvidena uporaba je ob zagonu aplikacije, kjer je pomembno, da video posnetkov ne predvajamo, preden so naloženi.

- **Set Screen Visible**: Akcija se uporablja za skrivanje osnovnega uporabniškega vmesnika in prikazovanje želenega okna uporabniškega vmesnika. Lahko se uporabi tudi obratno – skrije želeno okno uporabniškega vmesnika in pokaže osnovni uporabniški vmesnik. Predvidena uporaba akcije je za prikazovanje informacij, kot so navodila na začetku igre in rezultati na koncu igre.

- **Play Audio Clip One Shot**: Predvaja eno instanco zvočnega efekta. Akcija je uporabna v primerih, ko želimo predvajati neki dodaten zvočni efekt, ki ni prisoten v video posnetku, na primer aplavz ob novem najdaljšem izdihu.

- **Play Background Audio**: Akcija predvaja določen zvočni posnetek neodvisno od predvajanja video posnetkov in omogoča ponavljanje zvočnega posnetka. Akcija je uporabna v primerih, ko želimo, da ima celotna aplikacija neko glasbo ozadja, ki ni odvisna od predvajanja video posnetkov.

- **Wait For Headset Connection**: Akcija počaka, da uporabnik priklopi mikrofon. Predvidena uporaba akcije je na začetku aplikacije, ko želimo zagotoviti, da ima uporabnik priklopljen mikrofon, ki je potreben za pravilno delovanje aplikacije.

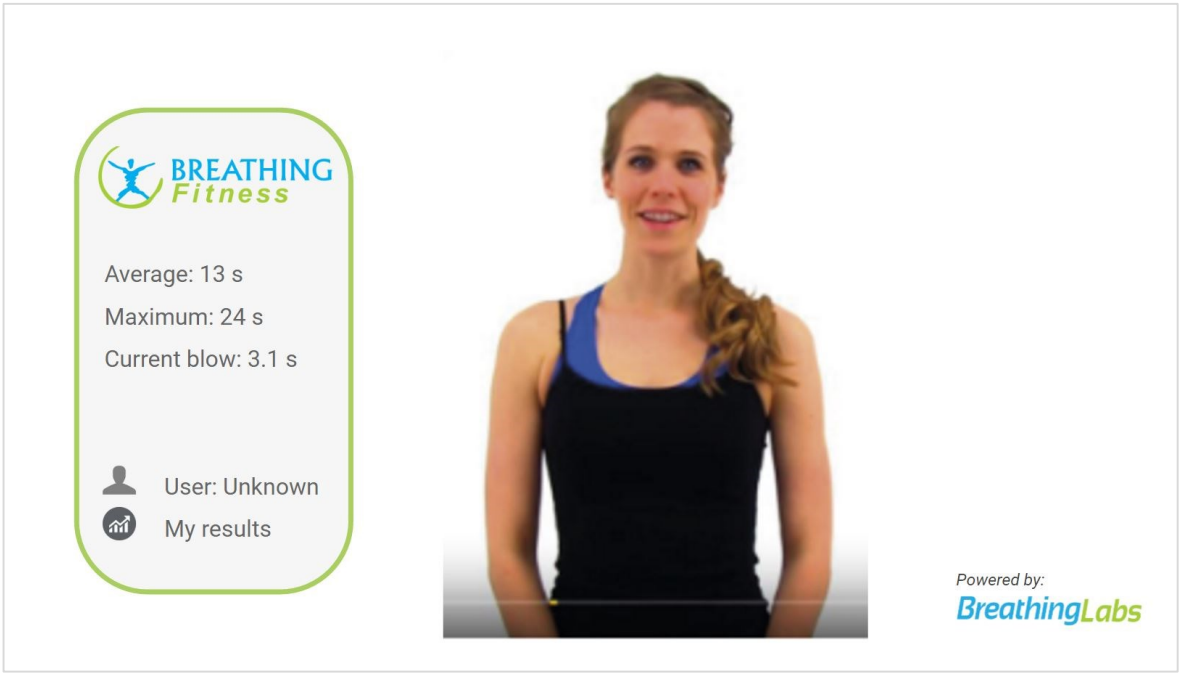
Dogodki, ki jih proži naš algoritem za zaznavanje dihanja, niso združljivi z dogodki, ki jih orodje Playmaker uporablja za prehajanje med stanji FSM. Zato smo razvili dodatno skripto, ki je naročena na dogodke zaznavanja dihanja in zna prožiti ustrezne dogodke orodja Playmaker na pripadajočih avtomatih FSM.

4.3 Uporabniški vmesnik

Naša orodja bodo uporabljena tako za razvoj aplikacij, ki bodo namenjene odraslim, kot za razvoj iger in aplikacij, ki bodo namenjene otrokom. Zato smo oblikovali dve različici uporabniškega vmesnika.

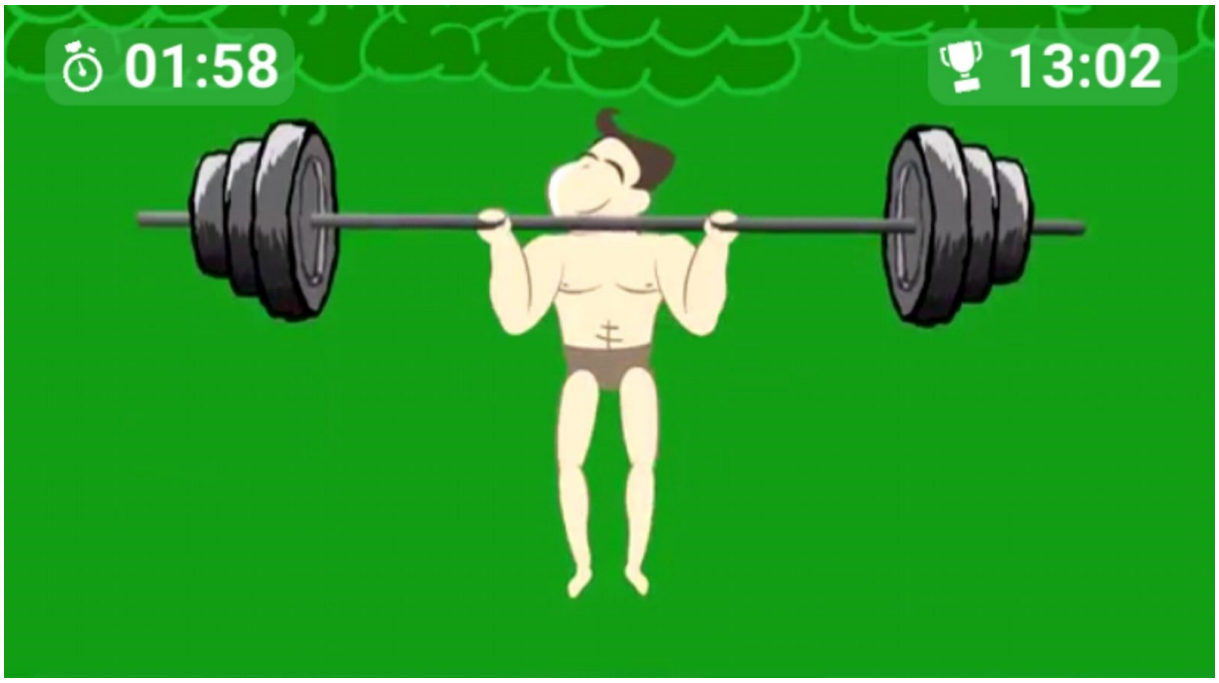
Prvi uporabniški vmesnik (Slika 4.4) je sestavljen iz okna, v katerem se nahaja logotip podjetja, prikaz povprečja dolžin izdiha, prikaz dolžine najdaljšega izdiha in prikaz dolžine trenutnega izdiha. Spodaj je prav tako dodeljen prostor za prikaz imena trenutnega uporabnika in gumb za prikaz podrobnejših statistik dihanja.

Oknu prvega uporabniškega vmesnika lahko uporabnik z vlečenjem (*angl. drag-and-drop*) poljubno spreminja pozicijo na zaslonu. Ker okno zasede razmeroma veliko površino zaslona, ga lahko uporabnik minimizira z dotikom oziroma klikom na logotip podjetja.



Slika 4.4 Uporabniški vmesnik za aplikacije, namenjene odraslim

Drugi uporabniški vmesnik (Slika 4.5) je sestavljen iz dveh nepremičnih oken v zgornjem levem in zgornjem desnem kotu. Okno v zgornjem levem kotu prikazuje dolžino trenutnega izdiha, okno v zgornjem desnem kotu pa prikazuje dolžino najdaljšega izdiha. Obe okni sta zasidrani na pripadajoč kot zaslona, tako da je njun odmik od roba zaslona vedno enak, ne glede na slikovno razmerje zaslona.



Slika 4.5 Uporabniški vmesnik za aplikacije, namenjene otrokom

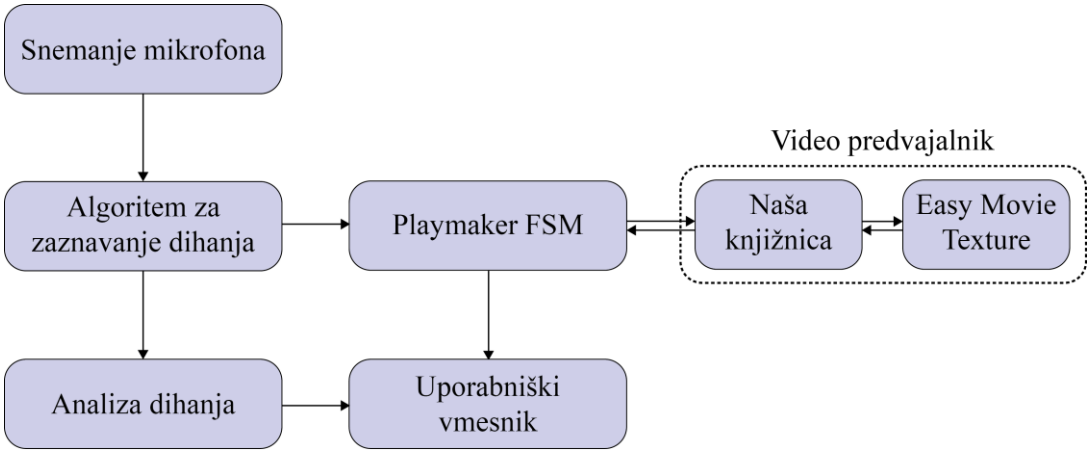
Za računanje podatkov, ki se prikazujejo na uporabniškem vmesniku, smo razvili preprosto orodje za analizo dihanja, ki je naročeno na dogodke zaznavanja dihanja. Orodje ob začetku izdiha zažene števec trajanja trenutnega izdiha in v vsakem okvirju na uporabniškem vmesniku posodobi prikaz trajanja trenutnega izdiha. Ob koncu izdiha se števec ustavi in se za nadaljnje računanje uporablja kot trajanje zadnjega izdiha. Izračuna se novo povprečno trajanje izdiha in novo največje trajanje izdiha. Na uporabniškem vmesniku nato posodobi obe vrednosti in jih shrani v spomin. Ob naslednjem zagonu aplikacije iz spomina prebere povprečno in najdaljše trajanje izdiha ter ustrezno nastavi začetne vrednosti uporabniškega vmesnika.

4.4 Pregled delovanja predvajalnika video posnetkov z uporabo dihanja

Algoritem za zaznavanje dihanja pridobiva informacije iz mikrofona. S proženjem dogodkov dihanja povzroča spremembe stanja avtomata Playmaker FSM, prav tako pa analizo dihanja obvešča o spremembah dihanja. Analiza dihanja prejete podatke analizira in statistiko dihanja prikazuje preko uporabniškega vmesnika.

Video predvajalnik je sestavljen iz dveh komponent – orodja Easy Movie Texture in naše knjižnice za nadzor predvajanja video posnetkov. Knjižnica prejema navodila za predvajanje

video posnetkov iz avtomata Playmaker FSM in na njihovi podlagi nadzoruje delovanje orodja Easy Movie Texture, hkrati pa avtomatu Playmaker FSM vrača podatke o stanju video posnetkov. Avtomat Playmaker FSM poleg upravljanja delovanja video predvajalnika upravlja še s prikazom obvestil na uporabniškem vmesniku. Shema povezav med vsemi komponenti je predstavljena na sliki 4.6.



Slika 4.6 Shema povezave komponent

Poglavje 5 Testiranje algoritma za zaznavanje dihanja

V tem poglavju bomo opisali postopek testiranja svojega algoritma za zaznavanje dihanja. Najprej bomo opisali vse testirane različice algoritma. Predstavili bomo svoje hipoteze in parametre testiranja. Opisali bomo postopek pridobivanja testnih podatkov in pojasnili potek testiranja. Na koncu bomo predstavili rezultate testiranja in jih ovrednotili.

5.1 Testirani algoritmi

Testirali smo štiri različice svojega algoritma, ki predstavljajo ključne točke v poteku razvoja končnega algoritma – vsak algoritem predstavlja nadgradnjo prejšnjega algoritma. Prvi testirani algoritem uporablja za zaznavanje dihanja enojni prag (3.1), drugi testirani algoritem uporablja dvojni prag (3.2), tretji testirani algoritem uporablja dvojni prag in minimizacijo glasnosti (3.3), četrti testirani algoritem pa uporablja tako dvojni prag kot tudi minimizacijo glasnosti in frekvenčno analizo signala (3.4).

5.2 Hipoteze

– Prvi algoritem:

  - Domnevamo, da bo algoritem z veliko natančnostjo zaznal tako začetke kot tudi konce izdihov. Hkrati domnevamo, da bo prišlo do večjega števila napačnih zaznavanj izdihov tako med govorom kot tudi med dihanjem.

  - Domnevamo, da bo algoritem občutljiv na presluh, in se mu bo zmanjšala splošna natančnost zaznavanja bodisi zaradi povečanega števila napačnih zaznavanj izdihov bodisi zaradi zmanjšanja pravilnega zaznavanja izdihov.

– Drugi algoritem:

  - Domnevamo, da bo algoritem z veliko natančnostjo zaznal tako začetke kot tudi konce izdihov. Domnevamo, da do napačnih zaznavanj izdihov med dihanjem ne bo prihajalo, med govorom pa bo še vedno nekaj napačnih zaznavanj, ampak manj kot pri prvem algoritmu.

- Domnevamo, da bo algoritem občutljiv na presluh in se mu bo zmanjšala splošna natančnost zaznavanja zaradi zmanjšanja pravilnega zaznavanja izdihov.

– Tretji algoritem:

- Domnevamo, da bo algoritem z veliko natančnostjo zaznal tako začetke kot tudi konce izdihov. Domnevamo, da do napačnih zaznavanj med dihanjem ne bo prihajalo, med govorom pa bo še vedno nekaj napačnih zaznavanj, ampak občutno manj kot pri drugem algoritmu.
- Domnevamo, da algoritem ne bo občutljiv na presluh in se mu v primeru presluha natančnost zaznavanja ne bo zmanjšala.

– Četrti algoritem:

- Domnevamo, da bo algoritem z veliko natančnostjo zaznal tako začetke kot tudi konce izdihov. Domnevamo, da do napačnih zaznavanj ne bo prihajalo niti med dihanjem niti med govorom.
- Domnevamo, da algoritem ne bo občutljiv na presluh in se mu v primeru presluha natančnost zaznavanja ne bo zmanjšala.

– Splošne hipoteze:

- Domnevamo, da bo vsak algoritem deloval s podobno natančnostjo neodvisno od posnetega prostovoljca na testiranem posnetku.

5.3 Parametri testiranja

Testirali bomo natančnost zaznavanja začetkov in koncev izdihov vseh štirih algoritmov. Prav tako bomo testirali odpornost algoritmov na govor in na motnjo presluha. Zanima nas tudi, kako dobro deluje algoritem na glasovih različnih oseb.

5.4 Testni podatki

Za namene testiranja smo posneli dihanje in govor štirih prostovoljcev. Posnetek vsakega prostovoljca sestoji iz dveh minut dihanja z uporabo tehnike PLB in iz glasnega branja vnaprej definiranega besedila, ki traja od minute do dveh minut.

Vsakemu posnetku smo z uporabo orodja Audio Degradation Toolbox [20] dodali glasbo v obliki dodanega šuma z razmerjem signal/šum (*angl. signal-to-noise ratio*) 10:1 [21]. Tako smo simulirali motnje mikrofona, do katerih prihaja zaradi presluha. Uporabljena glasba je bila The Show Must Be Go avtorja Kevina MacLeoda [22].

Za snemanje smo uporabili program Audacity in slušalke z mikrofonom Breathing+ Headset. Snemanje vseh podatkov je potekalo v popolni tišini in z uporabo enakih nastavitev na istem računalniku za vse štiri prostovoljce. V nadaljevanju bo vsak posnetek označen z imenom prostovoljca in oznako dodane motnje, na primer: posnetek *Tilen* predstavlja originalen posnetek prvega prostovoljca, posnetek *Tilen_C* pa predstavlja posnetek prvega prostovoljca z dodano motnjo.

5.5 Potek testiranja

Za vsak posnetek smo subjektivno določili točke začetkov in koncev izdihov. Določili smo tudi območje dovoljene napake, ki nam pove, s kakšno zamudo ali koliko prehitro lahko algoritem zazna začetek oziroma konec izdiha, da se bo detekcija še vedno smatrala kot pravilna.

Snemanje vhoda na mikrofonu smo izključili in ga nadomestili s svojimi testnimi podatki, tako da je rezultat algoritma z uporabo enakih parametrov na istem zvočnem posnetku vedno enak.

Vse posnetke smo spustili skozi algoritem in rezultate shranili v obliki XML. Nato smo rezultate algoritma primerjali s prej določenimi pravilnimi rezultati. Če je algoritem zaznal začetek oziroma konec izdiha znotraj območja dovoljene napake, potem smo zaznano točko opredelili kot resnično pozitivno (*angl. true positive*). Če je algoritem točko zaznal izven območja dovoljene napake, potem smo zaznano točko opredelili kot napačno pozitivno (*angl. false positive*). Če je algoritem znotraj območja dovoljene napake zaznal dve ali več točk, potem smo eno opredelili kot resnično pozitivno, ostale pa kot napačno pozitivne. Če algoritem znotraj območja dovoljene napake ni zaznal nobene točke, potem smo rezultat opredelili kot napačno negativen (*angl. false negative*).

Za vsak posnetek in njegove različice smo nato izračunali preciznost (*angl. precision*), priklic (*angl. recall*) in mero F (*angl. F-measure*) [23]. Ko smo imeli vse rezultate, smo izračunali še skupno preciznost, priklic ter mero F za vsak algoritem.

Preciznost predstavlja razmerje med številom pravilno zaznanih rezultatov in številom vseh zaznanih rezultatov. Pove nam, kakšna je verjetnost, da je zaznan rezultat pravilen. Računa se po formuli (5.1).

$$Precision = \frac{tp}{tp + fp} \tag{5.1}$$

Priklic predstavlja razmerje med številom pravilno zaznanih rezultatov in številom vseh pravilnih rezultatov. Pove nam, kakšen delež vseh pravilnih rezultatov smo pravilno zaznali. Računa se po formuli (5.2).

$$Recall = \frac{tp}{tp + fn} \tag{5.2}$$

Mera F (*ang. F-Measure*) predstavlja harmonično sredino preciznosti in priklica. V primerjavi z aritmetično sredino teži proti manjšim vrednostim, zato je vpliv manjših vrednosti nanjo večji kot vpliv večjih vrednosti. Računa se po formuli (5.3).

$$F\_Measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.3}$$

5.6 Rezultati testiranja

	Začetek izdiha			*Konec izdiha*		
Posnetek	**Preciznost**	**Priklic**	**Mera F**	**Preciznost**	**Priklic**	**Mera F**
Tilen	0.081	1.000	0.151	0.081	1.000	0.151
Maja	0.786	1.000	0.880	0.786	1.000	0.880
Luka	0.025	1.000	0.050	0.025	1.000	0.050
Matevž	0.031	1.000	0.061	0.031	1.000	0.061
Tilen_C	0.048	1.000	0.091	0.048	1.000	0.091
Maja_C	0.400	1.000	0.571	0.400	1.000	0.571
Luka_C	0.023	1.000	0.046	0.023	1.000	0.046
Matevž_C	0.029	1.000	0.056	0.029	1.000	0.056
Skupno	**0.075**	**1.000**	**0.139**	**0.075**	**1.000**	**0.139**

Tabela 5.1 Rezultati algoritma z enojnim pragom

	Začetek izdiha			*Konec izdiha*		
Posnetek	**Preciznost**	**Priklic**	**Mera F**	**Preciznost**	**Priklic**	**Mera F**
Tilen	0.153	1.000	0.265	0.153	1.000	0.265
Maja	0.957	1.000	0.978	0.957	1.000	0.978
Luka	0.036	1.000	0.069	0.036	1.000	0.069
Matevž	0.042	1.000	0.080	0.042	1.000	0.080
Tilen_C	0.112	1.000	0.202	0.112	1.000	0.202
Maja_C	0.917	1.000	0.957	0.917	1.000	0.957
Luka_C	0.035	1.000	0.068	0.035	1.000	0.068
Matevž_C	0.043	1.000	0.082	0.043	1.000	0.082
Skupno	**0.122**	**1.000**	**0.218**	**0.122**	**1.000**	**0.218**

Tabela 5.2 Rezultati algoritma z dvojnim pragom

	Začetek izdiha			***Konec izdiha***		
Posnetek	**Preciznost**	**Priklic**	**Mera F**	**Preciznost**	**Priklic**	**Mera F**
Tilen	1.000	1.000	1.000	1.000	1.000	1.000
Maja	1.000	1.000	1.000	1.000	1.000	1.000
Luka	0.833	1.000	0.909	0.833	1.000	0.909
Matevž	1.000	1.000	1.000	1.000	1.000	1.000
Tilen_C	0.917	1.000	0.957	0.917	1.000	0.957
Maja_C	1.000	1.000	1.000	1.000	1.000	1.000
Luka_C	1.000	1.000	1.000	1.000	1.000	1.000
Matevž_C	1.000	1.000	1.000	1.000	1.000	1.000
Skupno	**0.977**	**1.000**	**0.988**	**0.977**	**1.000**	**0.988**

Tabela 5.3 Rezultati algoritma z dvojnim pragom in minimizacijo glasnosti

	Začetek izdiha			***Konec izdiha***		
Posnetek	**Preciznost**	**Priklic**	**Mera F**	**Preciznost**	**Priklic**	**Mera F**
Tilen	1.000	1.000	1.000	1.000	1.000	1.000
Maja	0.909	0.909	0.909	1.000	1.000	1.000
Luka	1.000	1.000	1.000	1.000	1.000	1.000
Matevž	0.750	0.750	0.750	1.000	1.000	1.000
Tilen_C	1.000	1.000	1.000	1.000	1.000	1.000
Maja_C	1.000	1.000	1.000	1.000	1.000	1.000
Luka_C	1.000	1.000	1.000	1.000	1.000	1.000
Matevž_C	0.750	0.750	0.750	1.000	1.000	1.000
Skupno	**0.952**	**0.952**	**0.952**	**1.000**	**1.000**	**1.000**

Tabela 5.4 Rezultati algoritma z dvojnim pragom, minimizacijo glasnosti in frekvenčno analizo

5.7 Vrednotenje rezultatov

Rezultati testiranja prvega algoritma z enojnim pragom potrjujejo naše začetne hipoteze. Algoritem je pravilno zaznal vse začetke in konce izdihov, kar lahko razberemo iz vrednosti priklica, ki je enaka 1.000. Prišlo je do večjega števila napačnih zaznav tako med dihanjem kot tudi med govorom, kar je skupno natančnost zaznavanja začetkov in koncev izdihov znižalo na 0.075. Če primerjamo rezultate posameznih posnetkov z in brez dodanega presluha, opazimo, da je bila v vseh primerih natančnost zaznavanja manjša pri posnetkih s presluhom. Čeprav so bili vsi začetki in konci izdihov zaznani pravilno, je algoritem neprimeren za uporabo, ker je število napačnih zaznav preveliko.

Rezultati testiranja drugega algoritma z dvojnim pragom prav tako potrjujejo vse naše začetne hipoteze. Vsi začetki in konci izdihov so bili pravilno zaznani, prišlo pa je do večjega števila napačnih zaznav. V tem primeru so se napačne zaznave pojavljale le med govorom in jih je bilo manj kot pri prvem algoritmu. Algoritem je na večini posnetkov s presluhom dosegel slabšo natančnost kot na posnetkih brez presluha. Skupna natančnost zaznavanja začetkov in koncev izdihov znaša 0.122, kar predstavlja 63% izboljšavo v primerjavi s prvim algoritmom. Natančnost algoritma je še vedno premajhna, da bi bil algoritem primeren za uporabo.

Rezultati testiranja tretjega algoritma z dvojnim pragom in minimizacijo glasnosti ponovno potrjujejo naše začetne hipoteze. Vsi začetki in konci izdihov so bili pravilno zaznani, do napačnih zaznavanj pa je prišlo le pri dveh od skupno osmih posnetkov. Vsa napačna zaznavanja so se pojavila med govorom. Pri posnetkih s presluhom je pri enem posnetku prišlo do zmanjšanja natančnosti zaznavanja, pri dveh se natančnost ni spremenila, pri enem pa se je celo povečala. Skupna natančnost zaznavanja začetkov in koncev izdihov znaša 0.977, kar predstavlja sedemkratno povečanje v primerjavi z natančnostjo drugega algoritma. Natančnost algoritma je sedaj dovolj visoka, da je algoritem primeren za uporabo.

Rezultati testiranja četrtega algoritma z dvojnim pragom, minimizacijo in frekvenčno analizo le delno potrjujejo naše hipoteze. Vsi začetki in konci izdihov so bili pravilno zaznani. Napačnih zaznavanj koncev izdiha ni bilo, prišlo pa je do nekaj napačnih zaznavanj začetkov izdiha, ki so se pojavili kot posledica zamud pri zaznavanju. Pri posnetkih s presluhom je bila natančnost zaznavanja enaka ali večja kot pri posnetkih brez presluha. Skupna natančnost zaznavanja začetkov izdihov znaša 0.952, kar predstavlja 2.6% zmanjšanje v primerjavi z natančnostjo zaznavanja začetkov izdihov tretjega algoritma. Skupna natančnost zaznavanja koncev izdihov znaša 1.000, kar predstavlja 2.4% povečanje v primerjavi z natančnostjo zaznavanja koncev izdihov tretjega algoritma. Natančnost algoritma je dovolj visoka, da je algoritem primeren za

uporabo.

Hipoteza, da bo algoritem deloval s podobno natančnostjo na vseh posnetkih neodvisno od testiranega prostovoljca, velja le na zadnjih dveh algoritmih. Pri prvih dveh algoritmih se je natančnost zaznavanja precej spreminjala. Razlog za to je slaba odpornost prvih dveh algoritmov na govor, kjer ne moremo natančno predvideti števila napačnih zaznavanj. Dodaten razlog leži v razliki pri številu izdihov posameznega prostovoljca. Ker vsi algoritmi dobro zaznavajo začetke izdihov, natančnost pa je odvisna od števila pravilno zaznanih izdihov, je bila izračunana natančnost algoritmov večja pri prostovoljcih z večjim številom izdihov.

Glede na rezultate testiranja smo se odločili, da bomo v video predvajalniku privzeto uporabljali četrti algoritem z dvojnim pragom, minimizacijo in frekvenčno analizo. Pri tem algoritmu so bili namreč vsi primeri napak pri zaznavanju neposredna posledica zamud pri zaznavanju. Pri uporabi video predvajalnika se ob zaznanih začetkih in koncih izdihov po navadi zamenja trenutno predvajan video posnetek. Edina posledica zamude pri zaznavanju je tako kasnejša menjava video posnetka. V primeru napačnega zaznavanja izdiha, na primer med govorom, pa pride do nepričakovanega menjanja video posnetkov, kar je za uporabnika precej bolj moteče kot zamuda pri menjavi video posnetka. Za uporabo sta sicer dovolj natančna oba algoritma, zato bomo razvijalcem omogočili možnost izbire želenega algoritma.

Rezultati testiranja niso popolnoma zanesljivi. Za boljšo zanesljivost testov bi morali algoritme testirati na večjem številu prostovoljcev in boljše definirati testne posnetke – namesto določanja časa dihanja bi bilo boljše določiti število izdihov. Tako bi algoritem pri vseh posnetkih deloval s podobno preciznostjo in bi delovanje na različnih prostovoljcih lažje primerjali med seboj.

Poglavje 6 Zaključek

V diplomski nalogi smo se ukvarjali z razvojem orodij za pomoč pri razvoju dihalnih aplikacij v igralnem pogonu Unity. V ta namen smo razvili algoritem za zaznavanje dihanja in njegovo robustnost pokazali s testiranjem. Razvili smo še orodje za predvajanje video posnetkov na mobilnih napravah in ga povezali z algoritmom za zaznavanje dihanja. Predstavili smo postopek razvoja orodij, njihovo delovanje in način uporabe.

Orodja razvijalcem omogočijo lažji pristop k razvoju dihalnih aplikacij, tako da lahko v prihodnosti pričakujemo porast števila dihalnih aplikacij. Na tak način bo lahko več ljudi spoznalo vadbo dihanja z uporabo tehnike PLB. Upamo, da bomo tako vsaj komu omogočili bolj sproščen način življenja brez stresa in anksioznosti.

V podjetju Zdrav Dih, d. o. o., se razvita orodja že uporabljajo za razvoj dihalnih aplikacij, ki bodo bolnike, ki trpijo zaradi bolezni dihal, motivirale k redni vadbi z uporabo tehnike PLB, njihovim zdravnikom pa bodo omogočile sledenje statistikam dihanja vsakega bolnika.

Med razvojem smo dosegli večino zastavljenih ciljev, a še vedno obstaja precej možnosti za nadaljnji razvoj. Hitrost zaznavanja dihanja bi lahko še izboljšali, tako da bi poiskali alternativo minimizaciji glasnosti, ki v zaznavanje vnaša latenco približno 200 ms. Orodja bi lahko naredili bolj dostopna z razvojem lastnega vtičnika za predvajanje video posnetkov. Tako bi se izognili potrebi po nakupu plačljivega orodja Easy Movie Texture in razvoj dihalnih aplikacij omogočili širši publiki.

Literatura

[1] M. Nield, G. Soo Hoo, J. Roper, S. Santiago, "Efficacy of Pursed-Lips Breathing: A breathing pattern retraining strategy for dyspnea reduction", *Journal of Cardiopulmonary Rehabilitation and Prevention*, avg. 2007, str. 237–244.

[2] COPD International – Pursed Lip Breathing. `http://copd-international.com/PLB.html`. [Online; Dostopano 31.8.2016].

[3] Unity – Game Engine. `https://unity3d.com`. [Online; Dostopano 20.8.2016].

[4] Unity – Multiplatform. `https://unity3d.com/unity/multiplatform`. [Online; Dostopano 20.8.2016].

[5] Unity – Store. `https://store.unity.com`. [Online; Dostopano 20.8.2016].

[6] Unity – Manual: Learning the Interface. `https://docs.unity3d.com/Manual/LearningtheInterface.html`. [Online; Dostopano 21.8.2016].

[7] Unity – Manual: Prefabs. `https://docs.unity3d.com/Manual/Prefabs.html`. [Online; Dostopano 21.8.2016].

[8] Unity Asset Store. `https://www.assetstore.unity3d.com`. [Online; Dostopano 22.8.2016].

[9] Playmaker – Asset Store. `https://www.assetstore.unity3d.com/en/#!/content/368`. [Online; Dostopano 22.8.2016].

[10] Unity – Manual: Movie Texture. `https://docs.unity3d.com/Manual/class-MovieTexture.html`. [Online; Dostopano 23.8.2016].

[11] Unity – Scripting API: Handheld.PlayFullScreenMovie. https://docs.unity3d.com/ScriptReference/Handheld.PlayFullScreenMovie.html. [Online; Dostopano 23.8.2016].

[12] Easy Movie Texture – Asset Store. https://www.assetstore.unity3d.com/en/#!/content/10032. [Online; Dostopano 23.8.2016].

[13] Unity – Manual: Primitive and Placeholder Objects. https://docs.unity3d.com/Manual/PrimitiveObjects.html. [Online; Dostopano 23.8.2016].

[14] Breathing labs – Breathing+ Headset. https://www.breathinglabs.com/breathing-technology/breathing-headset/. [Online; Dostopano 24.8.2016].

[15] Unity – Manual: Audio Source. https://docs.unity3d.com/Manual/class-AudioSource.html.[Online; Dostopano 26.8.2016].

[16] Unity – Scripting API: AudioClip. https://docs.unity3d.com/ScriptReference/AudioClip.html. [Online; Dostopano 26.8.2016].

[17] Unity – Manual: Audio Mixer. https://docs.unity3d.com/Manual/AudioMixer.html. [Online; Dostopano 26.8.2016].

[18] I. Škraba, "Vhodno-izhodne naprave 2: Lastnosti električnih povezav v digitalnih sistemih", (prosojnice) Fakulteta za računalništvo in informatiko, 2016, str. 47.

[19] D. A. Lyon, "The Discrete Fourier Transform, Part 4: Spectral Leakage", *Journal of object technology,* , št. 7, zv. 8, str. 23-34, nov. 2009.

[20] M. Mauch, S. Ewert, "The Audio Degradation Toolbox and its Application to Robustness Evaluation, Proceedings" , *14th International Society for Music Information Retrieval Conference*, 2013.

[21] Signal-to-noise ratio. https://en.wikipedia.org/wiki/Signal-to-noise_ratio. [Online; Dostopano 28.8.2016].

[22] K. MacLeod, "The Show Must Be Go", (incompetech.com), Licensed under Creative Commons: By Attribution 3.0 License, http://creativecommons.org/licenses/by/3.0/.

[23] Precision and Recall. https://en.wikipedia.org/wiki/Precision_and_recall. [Online; Dostopano 28.8.2016].